# Tools that Enable DevSecOps

**Stuart Cianos, CISSP**
scianos@alphavida.com
Security Architect @ Medallia

# Changes/Corrections:

- 2018-06-20: Corrected typo on page 55; term "variable" corrected to "tag"

# Disclaimer

The views and opinions expressed during this conference are those of the speakers and do not necessarily reflect the views and opinions held by the Information Systems Security Association (ISSA), the Silicon Valley ISSA, the San Francisco ISSA or the San Francisco Bay Area InfraGard Members Alliance (IMA). Neither ISSA, InfraGard, nor any of its chapters warrants the accuracy, timeliness or completeness of the information presented. Nothing in this conference should be construed as professional or legal advice or as creating a professional-customer or attorney-client relationship. If professional, legal, or other expert assistance is required, the services of a competent professional should be sought.

# Tools that enable DevSecOps

AKA *Moving Fast with Open Source Tools for Compliance in Sensitive Environments*

June 19, 2018 – Security in the Age of Disruption

# The Challenge

- Move *FAST*
  - Most teams are using various forms of agile development practices because of realized productivity gains as well as changes in technology.
    - Rapid release cycles and CI/CD
      - Is the code *and* infrastructure testable? Proveable?
        - How to deploy a test environment?
    - Infrastructure as Code (IAC)
      - Cloud environments are the norm, not the exception
        - Multiple providers, APIs
      - Virtualization and Containerization
        - Docker, Kubernetes, Mesos
      - SDN is *everywhere* - AWS, Azure, GCE, on-premise

# The Challenge

- Decentralization and Democratization
  - Most teams are using various forms of agile development practices because of realized productivity gains as well as changes in technology.
    - Devops changes the nature of infrastructure
      - The days of the "system administrator", "network administrator", etc. targeting specific platforms or base software (OS) configuration are over… or numbered depending on the organization/who you ask.
      - Devops = Development + Operations
        - Many different areas of development impacting what was originally the subject area of the sysadmin or network admin.

# The Challenge

- Infrastructure as Code
  - Infrastructure is no longer physical; it is logical and mutable
    - Infrastructure as Code (IAC)
      - How are changes to the environment being
        - Managed?
        - Reviewed?
        - Applied?
        - Logged?

# The Challenge

- Infrastructure as Code
  - Infrastructure is no longer physical; it is logical and mutable
    - Infrastructure as Code (IAC)
      - How are changes to the environment being
        - Managed?
          - *Where is the central configuration/code repository?*
        - Reviewed?
          - *How are code reviews documented?*
        - Applied?
          - *How can we detect changes, what *will* be changed?*
        - Logged?
          - *Are changes logged? Can we know the differences between the deployment a year ago vs. today?*

# The Challenge

YES WE CAN!

# The Tooling

- Infrastructure as Code
  - How are changes to the environment being
    - Managed?
      - *Where is the central configuration/code repository?*
        - GIT, as well as other SCMs
    - Reviewed?
      - *How are code reviews documented?*
        - Using GIT pull requests, Gerrit, GitLab, GitHub (proprietary), GOGS, etc.
    - Applied?
      - *How can we detect changes, what *will* be changed?*
        - Terraform, CloudFormation (proprietary, AWS), Ansible, etc.
    - Logged?
      - *Are changes logged? Can we know the differences between the deployment a year ago vs. today?*
        - GIT, as well as other SCMs

# The Tooling

- Additional tooling...
  - Building master operating system images
    - How are system images built for cloud environments?
      - Packer
        - Can pull and customize an existing AMI on AWS
        - Can use an existing ISO and build a clean image for eventual use on AWS.
          - An important consideration if you need to run your own clean-room, validated images in sensitive environments (as well as enclaves like AWS GovCloud)

# The Tooling

- Additional tooling...
  - Deploying infrastructure
    - How can infrastructure be defined and stored in an SCM?
      - Terraform
        - Defines infrastructure as declarative code
        - Can deploy on multiple cloud environments and on-premise environments.
          - Supports all the common ones like AWS, Google Compute, Azure…
        - Tracks the state of the deployment
          - Can determine what has changed between the current desired state defined by code and what is actually deployed
            - Can help track what will be changed for change management processes
          - Can provide the necessary data to determine what was deployed using Terraform and what objects were not

# The Tooling

- Infrastructure as Code
  - How are changes to the environment being
    - Managed?
      - *Where is the central configuration/code repository?*
        - GIT, as well as other SCMs
    - Reviewed?
      - *How are code reviews documented?*
        - Using GIT pull requests, Gerrit, GitLab, GitHub (proprietary), etc.
    - Applied?
      - *How can we detect changes, what *will* be changed?*
        - Terraform, CloudFormation (proprietary, AWS), Ansible, etc.
    - Logged?
      - *Are changes logged? Can we know the differences between the deployment a year ago vs. today?*
        - GIT, as well as other SCMs

# Coverage

- What are we specifically going to cover today?
  - GIT
    - Gitolite
    - Gerrit
      - Substitute Gitolite or Gerrit with Github, Gitlab, etc. if desired… the concepts and workflows are similar regardless of the tooling.
      - *Pro tip*: Gerrit was specifically designed for code review workflows, if code review is specifically at the top of your list...
  - Terraform
  - A very quick word about Packer

# GIT… for compliance!

- What is GIT?
  - A version control system, used to track changes over time to content.
  - A *distributed* version control system.
    - Well suited to distributed teams, unstable network connectivity. Every GIT checkout is a full copy of the SCM repository.
    - Performant across a variety of projects and workloads, tested and proven in the real world.
      - GIT is the SCM used for Linux Kernel Development
  - Has well-defined workflows covering code review, sign-off, and/or two-man controls in most any configuration or environment.

# GIT… workflows

- How to enforce a workflow
  - Bob, Alice, and Cindy work for a company
    - Bob is an engineer trying to commit code
    - Alice is the reviewer of Bob's code
    - Cindy is one of the senior engineers whom manages releases and enforces policies.
  - But how can workflows be enforced on a distributed VCS/SCM?
    - Releases are only built from the repo hosted by the company, not the copy on Bob's laptop
    - By controlling who can actually commit to the various repositories or *branches* of a repository, its possible to control, validate and sign what gets built through a defined process.

# GIT… workflows

- How can workflows be enforced on a distributed VCS/SCM?
  - Releases are only built from the copy of master or release branch stored on the SCM hosted by the company, not the distributed copy on Bob's laptop
    - When a release is built, it should be built through a documented and defined build pipeline.
      - The build pipeline usually performs some or all of the following tasks:
        - Compiles/validates the build by building various artifacts
        - The artifacts are then tested (i.e. unit tests, functional tests)
        - If tests succeed, the artifact is made available for deployment.
          - In a CD (continuous delivery/deployment) shop, the build pipeline may even push the release to production.

# GIT… workflows

- How can workflows be enforced on a distributed VCS/SCM?
  - By controlling who can actually commit to the various repositories or *branches* of a repository on the business' GIT host, its possible to control, validate and sign what gets built through a defined process.
    - ACME Widget Co. develops an application called "WidgetMaster"
      - WidgetMaster's source code is hosted on ACMEs GIT host
        - Developers must have their code peer reviewed
        - Developers build code in their local GIT repo, in a development branch
        - Developers push code in their development branch to ACME's GIT host
        - The code reviewer examines the code in the development branch and approves/denies/requests more changes
          - *and then...*

# GIT… workflows

- How can workflows be enforced on a distributed VCS/SCM?
  - By controlling who can actually commit to the various repositories or *branches* of a repository on the business' GIT host, its possible to control, validate and sign what gets built through a defined process.
    - ACME Widget Co. develops an application called "WidgetMaster"
      - WidgetMaster's source code is hosted on ACMEs GIT host
        - The code reviewer examines the code in the development branch and approves/denies/requests more changes
          - *and then… depending on the release model...*
            - The branch can be merged into master by the reviewer, *or*
            - The branch can be merged into master by the developer, after the reviewer adds their blessing, *or*
            - The reviewer sends a PR to the release manager for inclusion, *or*
            - Software like Gerrit is used to define the process and add an interface specifically targeting code reviews and management tracking.

# GIT… workflows

- How can workflows be enforced on a distributed VCS/SCM?
  - Irregardless of the workflow, the concepts are the same
    - Access is controlled to repositories as a whole, or perhaps branches.
    - The organization can use a variety of configurations to achieve the same controls.
    - Some of the controls will be defined by the software chosen, but most are up to the organization to choose.
      - Examples:
        - Gitolite is well suited to using branches for access control
        - Github/Gitlab are also well suited to using branches
        - Gerrit doesn't use branching, but uses references instead
        - Plain GIT request-pull doesn't use branching

# GIT… workflows

- How can workflows be enforced on a distributed VCS/SCM?
  - Irregardless of the workflow, the concepts are the same
    - Access is controlled to repositories as a whole, or perhaps branches.
    - The organization can use a variety of configurations to achieve the same controls.
    - Some of the controls will be defined by the software chosen, but most are up to the organization to choose.
      - Examples:
        - Gitolite is well suited to using branches for access control
        - Github/Gitlab are also well suited to using branches
        - Gerrit doesn't use branching, but uses references instead
        - Plain GIT request-pull doesn't use branching
      - Pro-tip: Pick the tool that works best for your org *and* developers!

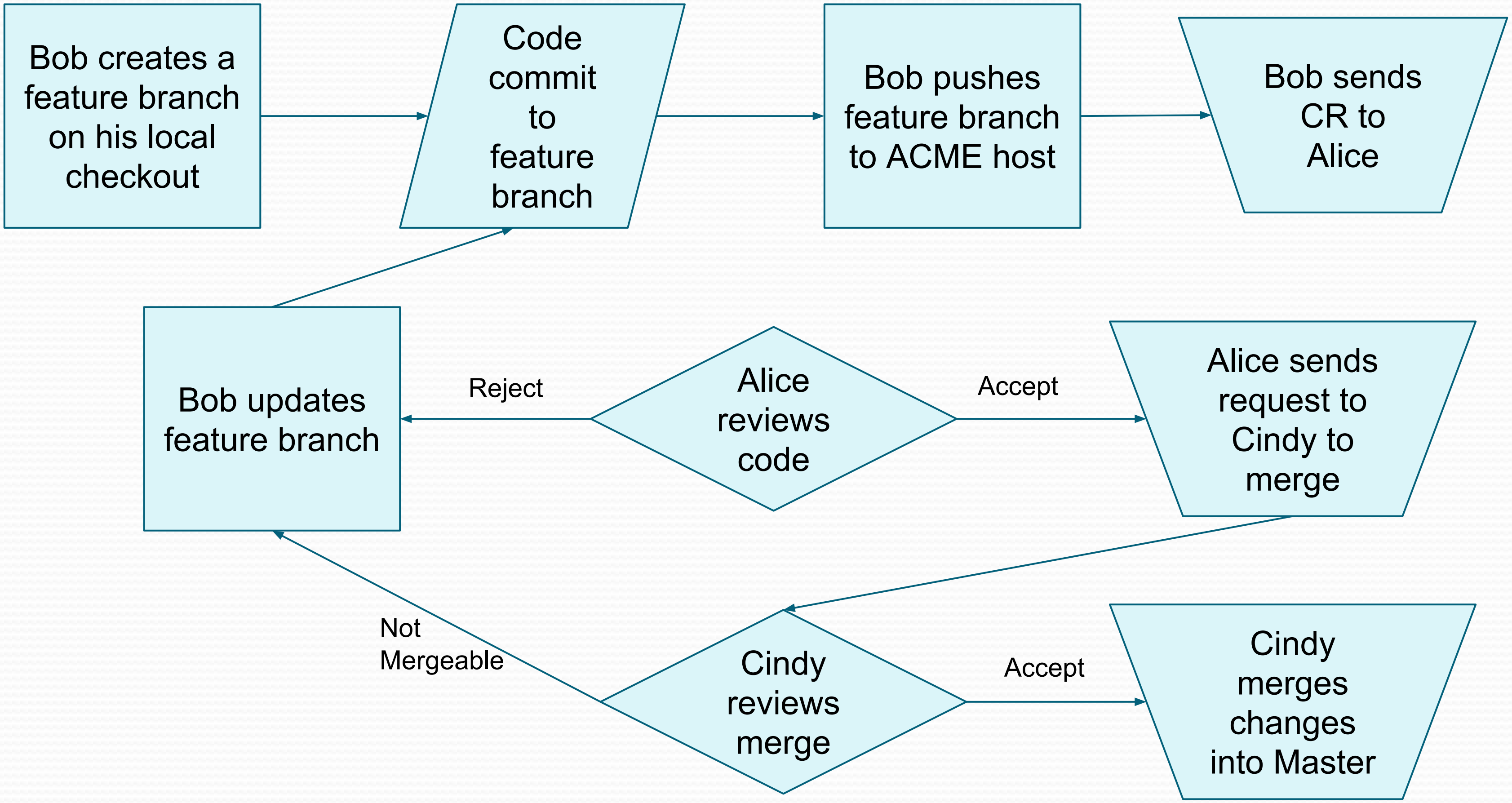
# GIT… workflows

- For ACME, the organization wants:
  - **To build their releases rolling off master**
  - An easy to maintain and install tool that allows them to enforce a workflow on top of their plain GIT host without changing the rest of their SCM infrastructure
  - To control access to some branches (like RO to MASTER for most)
- For ACME, the developers want:
  - A tool that stays out of their way and lets them use their existing workflows (change adverse)
  - Don't care about the GUI, they use the GIT CLI over SSH exclusively
- ACME has some options…
  - Gitolite? Gitlab? Github? …?
  - Move developers to Gerrit CR model?

# GIT… workflows

- For ACME, the organization decided on this workflow:
  - **Builds are a rolling release from the master branch**
    - Only Cindy (the release manager) can commit to master branch
  - Bob the developer creates a feature branch and makes changes
  - Bob commits changes back to his feature branch and pushes upstream to a feature branch on ACME's GIT host.
  - Alice reviews his change and approves/rejects
  - If approved, Alice notes the commit information and sends a request to Cindy to merge the code
  - Cindy merges the code into the Master branch.

# GIT… workflows



Bob creates a feature branch on his local checkout

Code commit to feature branch

Bob pushes feature branch to ACME host

Bob sends CR to Alice

Bob updates feature branch

Reject

Alice reviews code

Accept

Alice sends request to Cindy to merge

Not Mergeable

Cindy reviews merge

Accept

Cindy merges changes into Master

# GIT… workflows

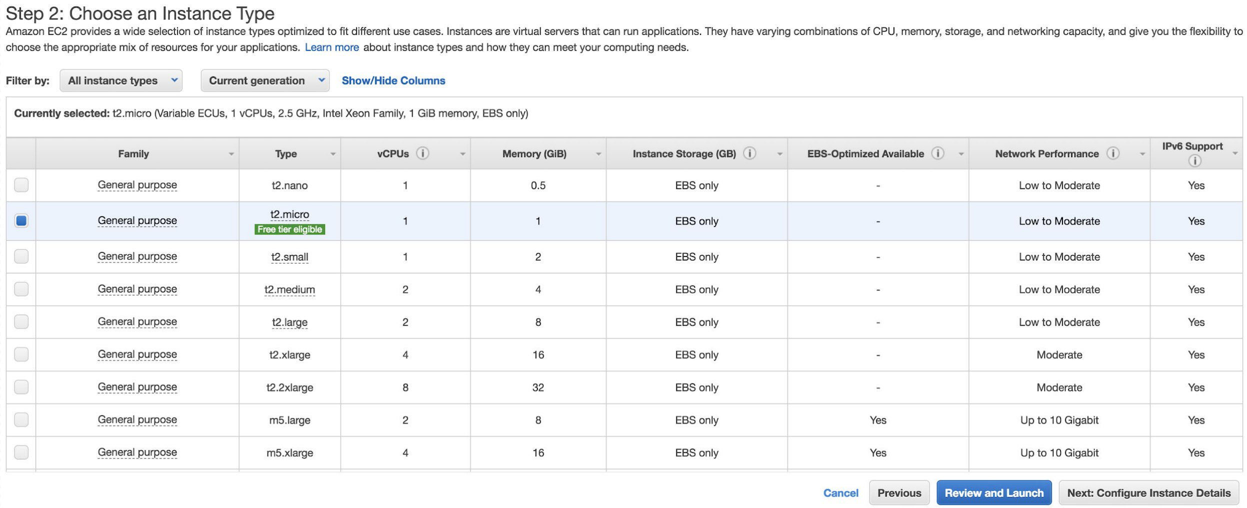
- For ACME, the organization decided on Gitolite:
  - Lightweight tool that supports the developer's use of GIT CLI and SSH.
  - Doesn't include a GUI or advanced code review facilities like Gerrit/Gitlab/Github, but controls access to branches (including master!)

Sample configuration for widgetmaster using groups "cindy_team" (for release manager team) and "bobs_team" (for development group)

```
repo widgetmaster
     RW+ master      =   @cindy_team # Rel Mgr
     -   master      =   @bobs_team  # Devs
     RW+             =   @bobs_team  # Devs
```

# Workflow defined…

- Now that ACME has defined their workflow, how can this workflow be used to define infrastructure?
  - How is infrastructure created?



Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. Learn more about instance types and how they can meet your computing needs.

Filter by: All instance types | Current generation

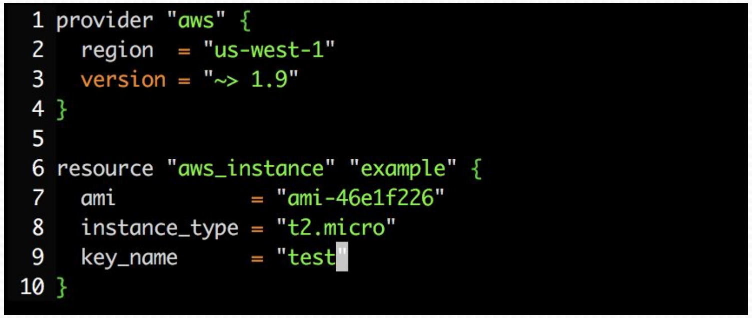
**Currently selected:** t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

| | Family | Type | vCPUs | Memory (GiB) | Instance Storage (GB) | EBS-Optimized Available | Network Performance | IPv6 Support |
|---|---|---|---|---|---|---|---|---|
| ☐ | General purpose | t2.nano | 1 | 0.5 | EBS only | - | Low to Moderate | Yes |
| ☑ | General purpose | t2.micro<br>Free tier eligible | 1 | 1 | EBS only | - | Low to Moderate | Yes |
| ☐ | General purpose | t2.small | 1 | 2 | EBS only | - | Low to Moderate | Yes |
| ☐ | General purpose | t2.medium | 2 | 4 | EBS only | - | Low to Moderate | Yes |
| ☐ | General purpose | t2.large | 2 | 8 | EBS only | - | Low to Moderate | Yes |
| ☐ | General purpose | t2.xlarge | 4 | 16 | EBS only | - | Moderate | Yes |
| ☐ | General purpose | t2.2xlarge | 8 | 32 | EBS only | - | Moderate | Yes |
| ☐ | General purpose | m5.large | 2 | 8 | EBS only | Yes | Up to 10 Gigabit | Yes |
| ☐ | General purpose | m5.xlarge | 4 | 16 | EBS only | Yes | Up to 10 Gigabit | Yes |

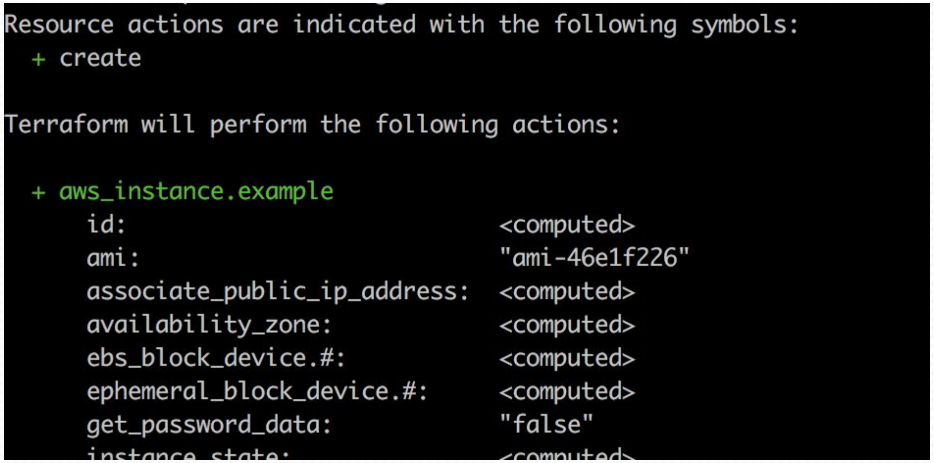
"Artisanal Infrastructure": Bob clicks on launch button and hopes that he is correct

# Workflow defined...

- Now that ACME has defined their workflow, how can this workflow be used to define infrastructure?
  - How is infrastructure created?



```
provider "aws" {
  region  = "us-west-1"
  version = "~> 1.9"
}

resource "aws_instance" "example" {
  ami           = "ami-46e1f226"
  instance_type = "t2.micro"
  key_name      = "test"
}
```



```
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  + aws_instance.example
      id:                           <computed>
      ami:                          "ami-46e1f226"
      associate_public_ip_address:  <computed>
      availability_zone:            <computed>
      ebs_block_device.#:           <computed>
      ephemeral_block_device.#:     <computed>
      get_password_data:            "false"
      instance_state:               <computed>
```

"Infrastructure as Code": Bob defines the host as code and can deploy automatically to any environment

# Workflow defined...

- Now that ACME has a GIT workflow and is defining their infrastructure as code, what does this mean?
  - If Bob wants to launch infrastructure, it goes through code review ahead of merge.
  - Bob can spin up and test the infrastructure on-the-fly in a test or staging environment he has access to, without manual intervention.
  - The team that deploys production now has a consistent infrastructure deployment process based on Terraform.
  - Alice has reviewed Bob's code, and the code that Alice reviewed is the code that Cindy merged into Master.
  - The deployment team deploys (only) the approved code.

# Workflow defined…

- Other important benefits
  - The infrastructure code can now be validated (or even tested!) as part of a CI/CD pipeline
  - The infrastructure can be easily re-deployed as part of a BC/DR process.
    - Major Plus!
  - The infrastructure can be easily deployed to multiple environments
    - Horizontal scaling, HA sites, and more!
  - The infrastructure deployment is:
    - Proveable
    - Reproducible
    - Documented **by default** as the end state is always defined!

# Organizing Terraform Projects

- Terraform does not force you into any particular organization of a project… **this is very much a *highly opinionated* guide!!!**
  - Early decisions can help or hurt your efforts
    - Pro-Tip: **DRY - Don't Repeat Yourself!** Design your project to enforce re-usability.
      - A project structure that uses modules to create reusable "pieces" or features (Terraform modules) is one recommended way to achieve this goal.
        - A Terraform module is a method for creating components that can be called from other components
      - Terraform also has support for "workspaces" which can be used to implement re-usability across environments, or a project can be designed to support it natively.

# Organizing Terraform Projects

- One recommended project structure…
  - Two top level entities
    - **Feature Modules**
      - A feature module implements functionality that can be deployed into environments.
      - A feature module is called upon by environment modules to deploy functionality in said environment
    - An environment, an environment ….
      - **Environment Modules**
        - An environment module defines which feature modules will be deployed.
        - Terraform state is tied to environment modules
        - An environment module lives in a specific environment

# Organizing Terraform Projects

- ACME has multiple environments; each environment is an AWS account for the purposes of this example...
  - Production 1 (HA)
  - Production 2 (HA)
  - Corporate Resources (HA)
  - Staging
  - Development

# Organizing Terraform Projects

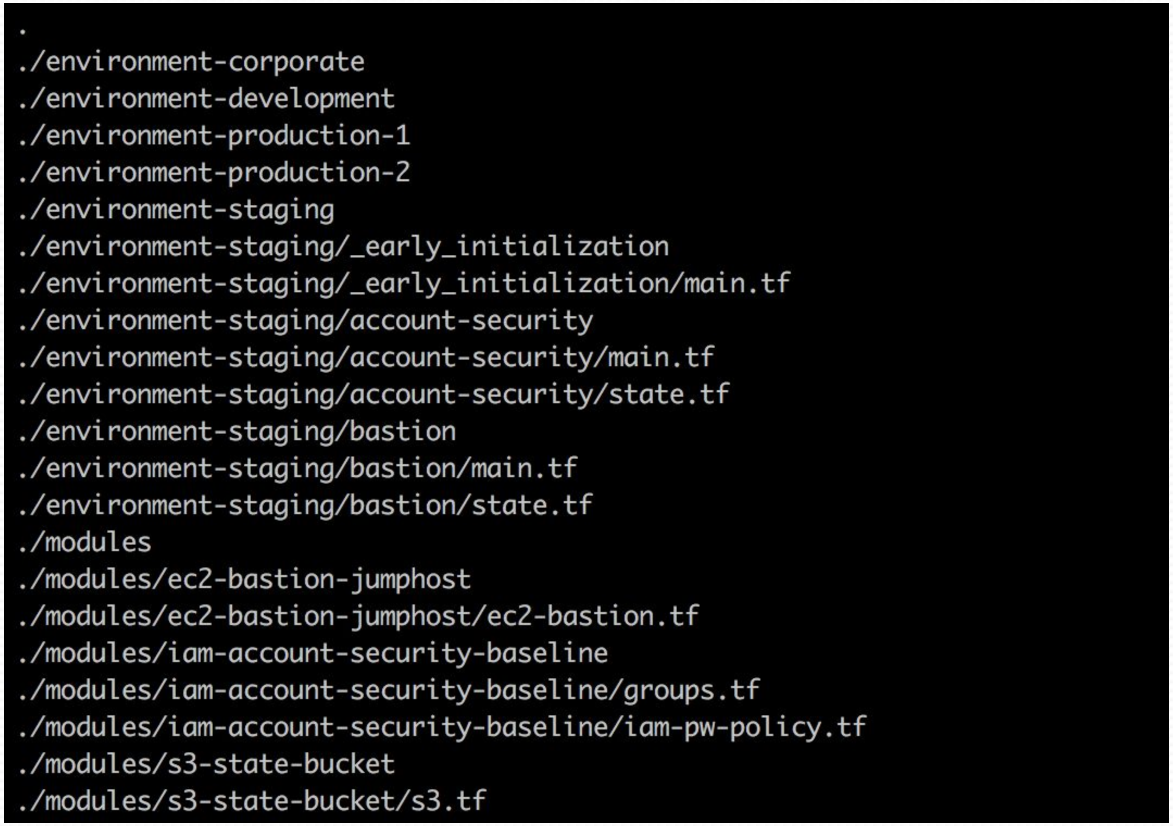
- ACME has multiple environments; each environment is an AWS account for the purposes of this example...
  - Production 1 (HA)
    - US-WEST-1
      - Availability Zone A
      - Availability Zone B
  - Production 2 (HA)
    - US-EAST-1
      - Availability Zone A
      - Availability Zone B
  - Corporate Resources (HA)
    - US-WEST-1
      - Availability Zone A
      - Availability Zone B
  - Staging
    - US-WEST-1
      - Availability Zone A
  - Development
    - US-WEST-1
      - Availability Zone A

# Organizing Terraform Projects

- ACME has three features they need deployed as part of their product...
  - An initial module which creates dependencies used to store Terraform state
  - A module which configures security properties on every account ACME manages, creates IAM groups, etc.
  - A module which creates a server for bastion access to the environment.
- The features must be re-usable across all environments.
  - The features/functionality is encapsulated in modules
  - Modules can (*should*!) be deployable to any environment (*if designed properly*!)

# Organizing Terraform Projects

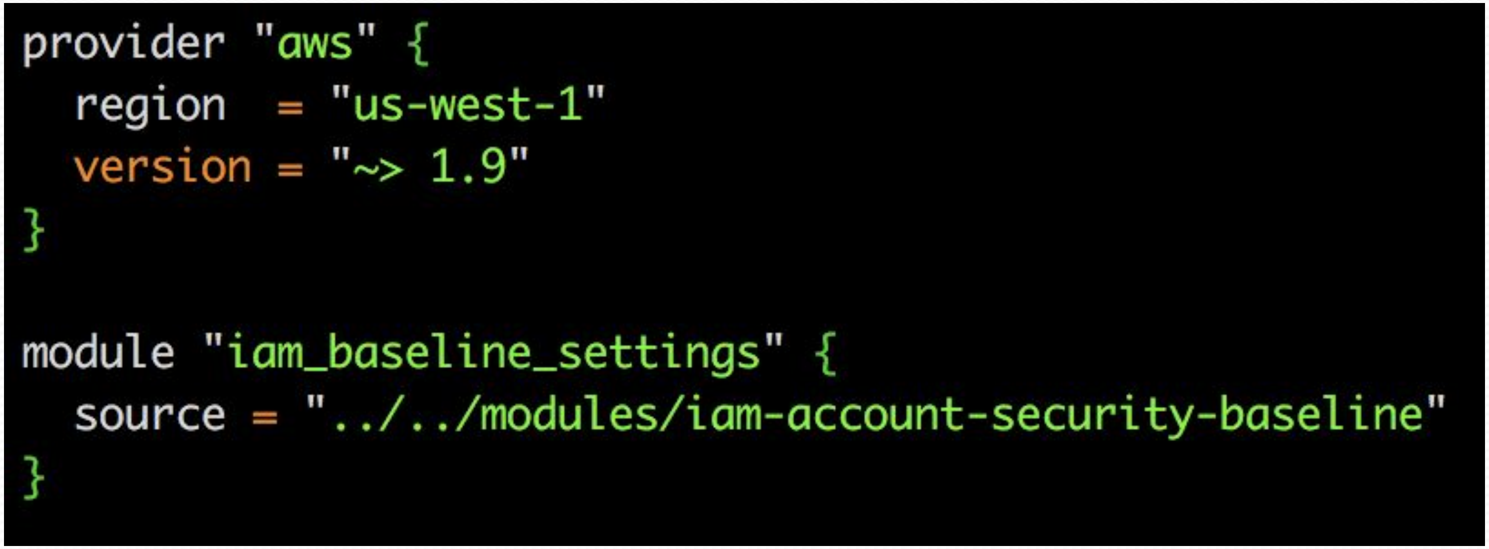
- Gives us the following project organization as a Terraform directory structure (w/ environment modules for staging):



```
.
./environment-corporate
./environment-development
./environment-production-1
./environment-production-2
./environment-staging
./environment-staging/_early_initialization
./environment-staging/_early_initialization/main.tf
./environment-staging/account-security
./environment-staging/account-security/main.tf
./environment-staging/account-security/state.tf
./environment-staging/bastion
./environment-staging/bastion/main.tf
./environment-staging/bastion/state.tf
./modules
./modules/ec2-bastion-jumphost
./modules/ec2-bastion-jumphost/ec2-bastion.tf
./modules/iam-account-security-baseline
./modules/iam-account-security-baseline/groups.tf
./modules/iam-account-security-baseline/iam-pw-policy.tf
./modules/s3-state-bucket
./modules/s3-state-bucket/s3.tf
```

# Organizing Terraform Projects

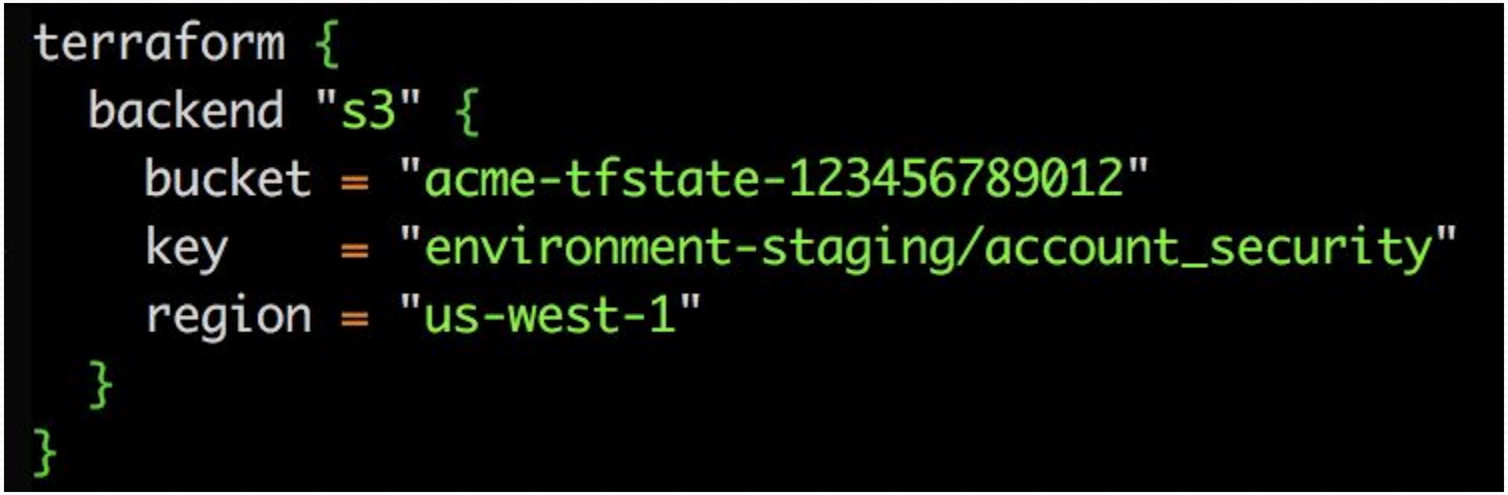
- Important highlights
  - Each environment module under an environment defines the provider(s) and state.
  - Each environment module picks the features it deploys by including the desired feature modules.



```
provider "aws" {
  region  = "us-west-1"
  version = "~> 1.9"
}

module "iam_baseline_settings" {
  source = "../../modules/iam-account-security-baseline"
}
```

# Organizing Terraform Projects

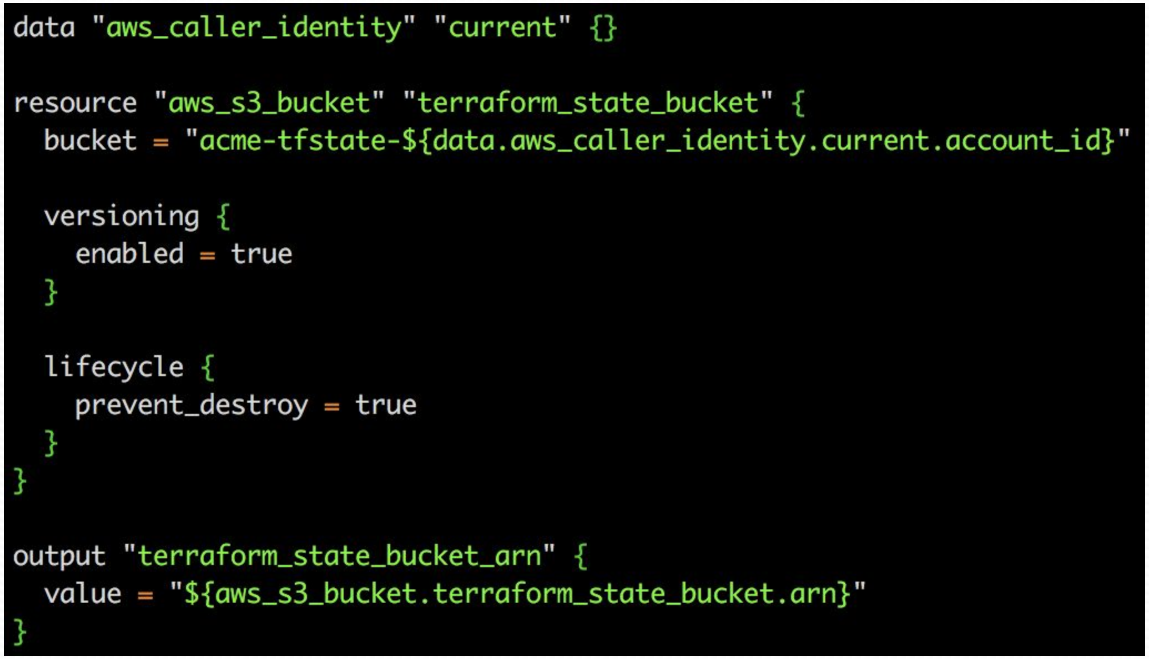
- Important highlights
  - Each environment module (except for _early_initialization) has a remote state configuration. For this example, remote state is stored in an S3 bucket associated with each account, and a key associated to each environment and module:



```
terraform {
  backend "s3" {
    bucket = "acme-tfstate-123456789012"
    key    = "environment-staging/account_security"
    region = "us-west-1"
  }
}
```

# Organizing Terraform Projects

- Important highlights
  - _early_initialization creates that bucket. This dependency is the first step of deployment with this methodology/project structure:



```
data "aws_caller_identity" "current" {}

resource "aws_s3_bucket" "terraform_state_bucket" {
  bucket = "acme-tfstate-${data.aws_caller_identity.current.account_id}"

  versioning {
    enabled = true
  }

  lifecycle {
    prevent_destroy = true
  }
}

output "terraform_state_bucket_arn" {
  value = "${aws_s3_bucket.terraform_state_bucket.arn}"
}
```

# Organizing Terraform Projects

- Important highlights
  - Note the use of variables to make some feature module re-usable (interpolation on the host's AWS name tag, S3 bucket name)
  - We aren't using tools like terragrunt for the purposes of this discussion
    - Terraform remote state configuration is managed by each environment module
      - State corruption or issues in one module won't impact another
    - Environment modules pass variables (parameters) to feature modules
    - You *may* want to consider TF workspaces or Terragrunt re: state management across environments. There are pros and cons…….

# Organizing Terraform Projects

- Important highlights
  - **State files *__may__* contain secrets**… understand how your code is creating resources.
    - For example, if you create a new private key in Terraform targeting AWS ACS the private key will be stored in state so that Terraform can convey the key. This may not be desired!
      - Define the empty resource and then import the public cert only as a safer alternative. Only the public certificate will be in state.
- Opportunity for enhancement!
  - The bucket created doesn't have encryption turned on by default.
    - **Explore the "encrypt=true" option available in Terraform re: S3 buckets and state (out of scope for this presentation)**

# Organizing Terraform Projects

- Important highlights
  - **State files *may* contain secrets**… cont'd…
    - This code snippet requires a private key to be stored in an adjacent file before deploy (probably not desired)
    - This code snippet also persists the private key in TF state

```
resource "aws_iam_server_certificate" "test_cert" {
  name_prefix      = "example-cert"
  certificate_body = "${file("self-ca-cert.pem")}"
  private_key      = "${file("test-key.pem")}"

  lifecycle {
    create_before_destroy = true
  }
}
```

# Organizing Terraform Projects

- Important highlights
  - **State files *may* contain secrets**… cont'd…
    - Since AWS does not allow private keys to be conveyed from an IAM certificate after creation/upload, the only way for Terraform to convey a key back to the caller after initial run is to maintain it in state.
      - Solution: Import the key securely to AWS, then import the reference back into Terraform
      - New empty resource would appear as follows in code:

```
resource "aws_iam_server_certificate" "test_cert" {
  certificate_body = ""
  private_key      = ""
}
```

# Organizing Terraform Projects

- Important highlights
  - **State files *<u>may</u>* contain secrets**… cont'd…
    - Now that we have an empty resource… import the certificate to AWS:

```
$ aws iam upload-server-certificate --server-certificate-name test1 --certificate-body
file://test.crt --certificate-chain file://ca.crt --private-key file://test.key
{
    "ServerCertificateMetadata": {
        "ServerCertificateName": "test1",
        ...
    }
}
```

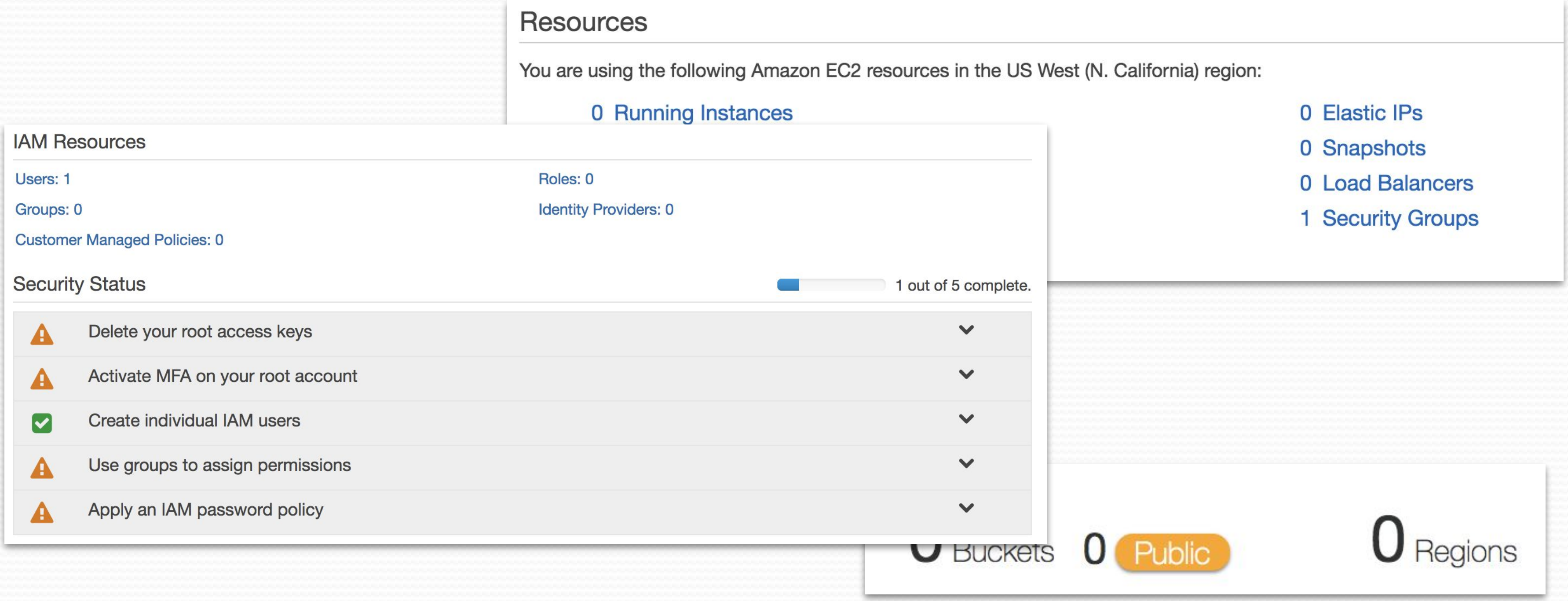
    - Tell Terraform about the resource:

```
$ terraform import module.test_certificate.aws_iam_server_certificate.test_cert test1
module.test_certificate.aws_iam_server_certificate.test_cert: Importing from ID
"test1"...
module.test_certificate.aws_iam_server_certificate.test_cert: Import complete!
  Imported aws_iam_server_certificate (ID: test1)
...
```

    - Now, only the **public cert** is in state and on the project's path

# Terraform in Action

- Time to deploy to staging!
  - **A fresh AWS account** with no configuration except the initial user I am presenting with...



Resources

You are using the following Amazon EC2 resources in the US West (N. California) region:

0 Running Instances

0 Elastic IPs

0 Snapshots

0 Load Balancers

1 Security Groups

IAM Resources

| | |
|---|---|
| Users: 1 | Roles: 0 |
| Groups: 0 | Identity Providers: 0 |
| Customer Managed Policies: 0 | |

Security Status — 1 out of 5 complete.

- ⚠ Delete your root access keys
- ✓ Create individual IAM users
- ⚠ Activate MFA on your root account
- ⚠ Use groups to assign permissions
- ⚠ Apply an IAM password policy

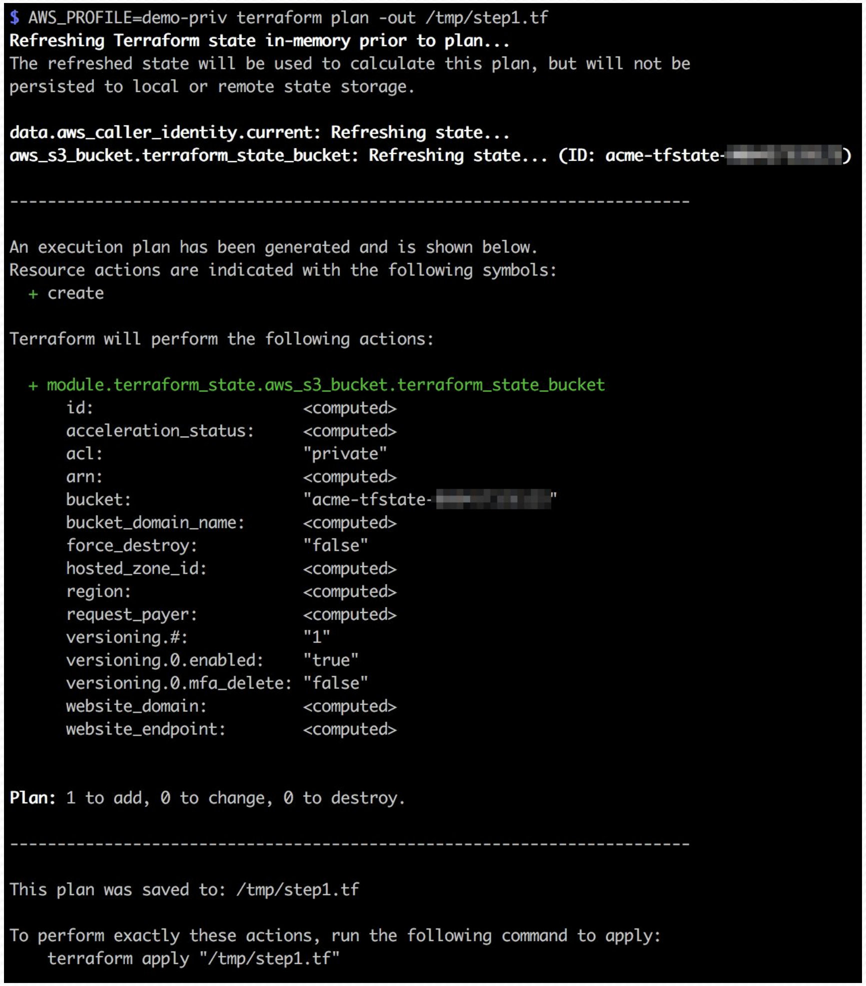
0 Buckets 0 Public 0 Regions

# Terraform in Action

- Time to deploy to staging!
  - **The deployment plan**
    - There are dependencies! Don't forget to document how to spin up environments given the configuration…
    - The environment module dependency order for this deployment to staging environment:
      - _early_initialization: Initialize the state bucket (one time only)
      - account-security: Set up IAM permissions and groups
      - bastion: Spin up a bastion host

# Terraform in Action

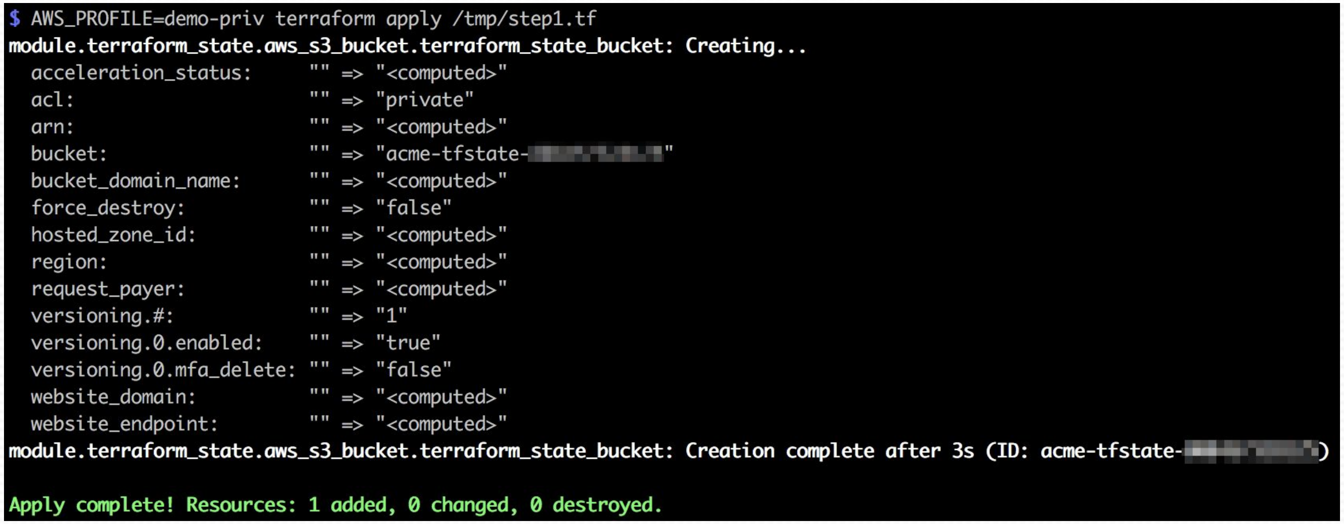
- Time to deploy to staging!
  - **Step 1:** _early_initialization
  - Since this is the first time spinning up the environment, a TF state bucket must be created.
  - **This is the only step where state is not remotely stored.**
  - **This single state file should be committed back to repo so that subsequent deployments are aware.**
  - All further state is remote.



```
$ AWS_PROFILE=demo-priv terraform plan -out /tmp/step1.tf
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

data.aws_caller_identity.current: Refreshing state...
aws_s3_bucket.terraform_state_bucket: Refreshing state... (ID: acme-tfstate-[redacted])

------------------------------------------------------------------------

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  + module.terraform_state.aws_s3_bucket.terraform_state_bucket
      id:                        <computed>
      acceleration_status:       <computed>
      acl:                       "private"
      arn:                       <computed>
      bucket:                    "acme-tfstate-[redacted]"
      bucket_domain_name:        <computed>
      force_destroy:             "false"
      hosted_zone_id:            <computed>
      region:                    <computed>
      request_payer:             <computed>
      versioning.#:              "1"
      versioning.0.enabled:      "true"
      versioning.0.mfa_delete:   "false"
      website_domain:            <computed>
      website_endpoint:          <computed>


Plan: 1 to add, 0 to change, 0 to destroy.

------------------------------------------------------------------------

This plan was saved to: /tmp/step1.tf

To perform exactly these actions, run the following command to apply:
    terraform apply "/tmp/step1.tf"
```

# Terraform in Action

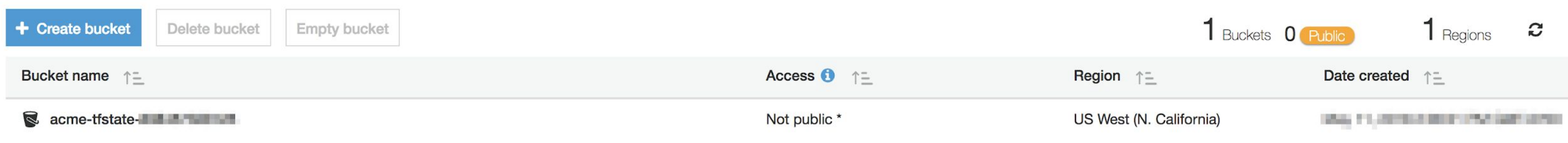
- Time to deploy to staging!
  - **Step 1:** _early_initialization
  - Notice how the <u>plan</u> told us exactly what changes are going to be made, and what the differences between the existing state and new state will be.
  - Apply the plan:



```
$ AWS_PROFILE=demo-priv terraform apply /tmp/step1.tf
module.terraform_state.aws_s3_bucket.terraform_state_bucket: Creating...
  acceleration_status:      "" => "<computed>"
  acl:                      "" => "private"
  arn:                      "" => "<computed>"
  bucket:                   "" => "acme-tfstate-[redacted]"
  bucket_domain_name:       "" => "<computed>"
  force_destroy:            "" => "false"
  hosted_zone_id:           "" => "<computed>"
  region:                   "" => "<computed>"
  request_payer:            "" => "<computed>"
  versioning.#:             "" => "1"
  versioning.0.enabled:     "" => "true"
  versioning.0.mfa_delete:  "" => "false"
  website_domain:           "" => "<computed>"
  website_endpoint:         "" => "<computed>"
module.terraform_state.aws_s3_bucket.terraform_state_bucket: Creation complete after 3s (ID: acme-tfstate-[redacted])

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

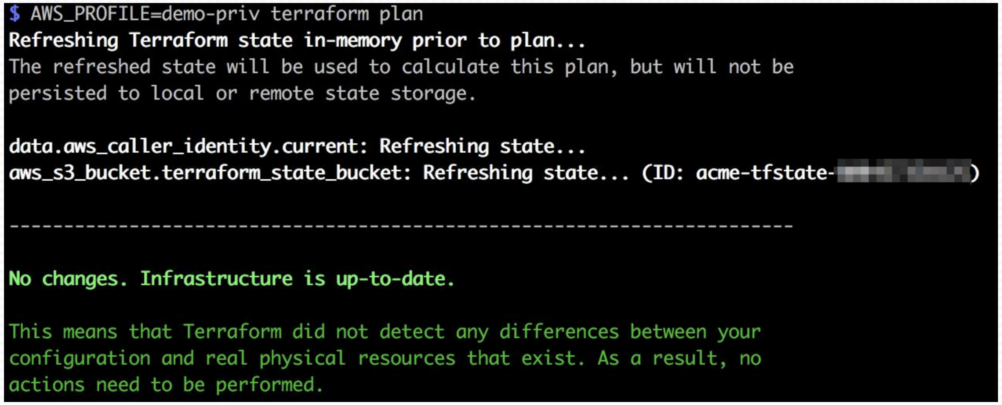
# Terraform in Action

- Time to deploy to staging!
  - **Step 1:** _early_initialization
  - The plan has been applied, and changes were made:



| Bucket name | Access | Region | Date created |
|---|---|---|---|
| acme-tfstate-[redacted] | Not public * | US West (N. California) | [redacted] |

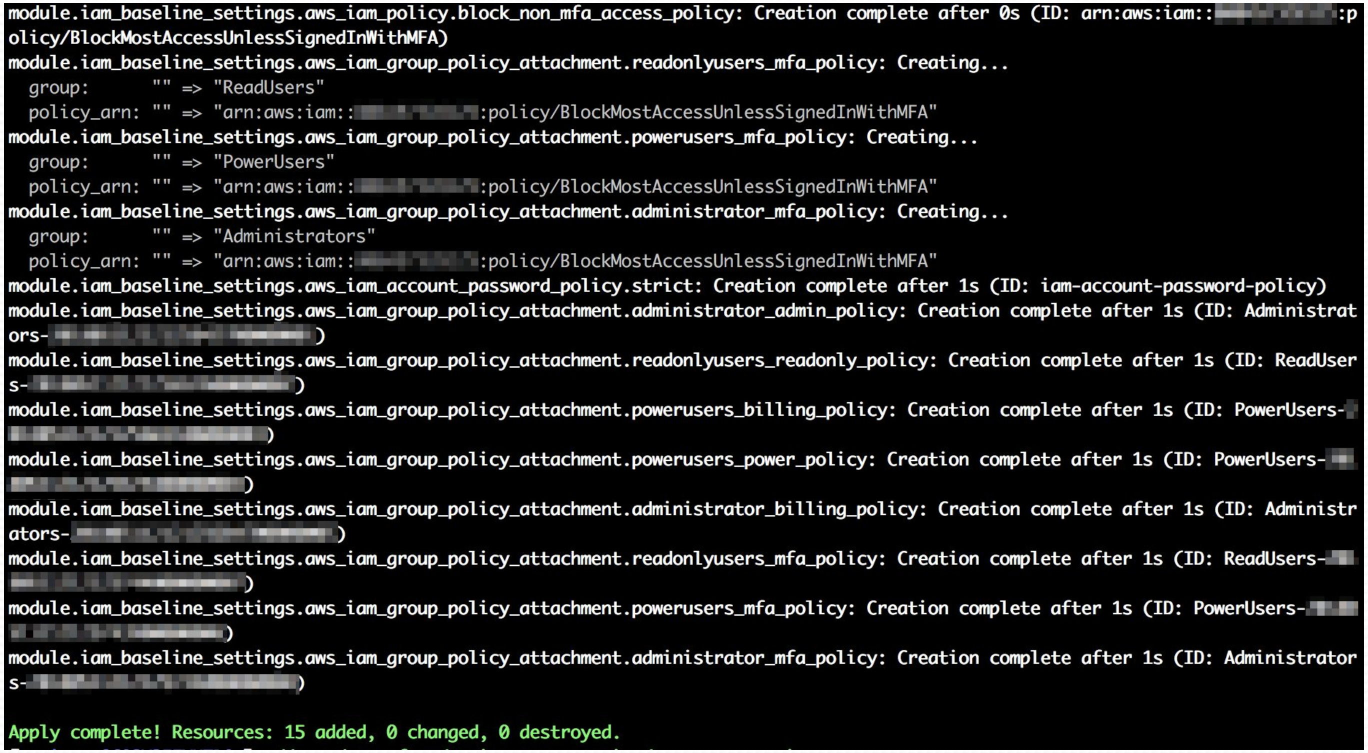
  - Re-running terraform plan or apply shows that its aware of the bucket on future deployments:



```
$ AWS_PROFILE=demo-priv terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

data.aws_caller_identity.current: Refreshing state...
aws_s3_bucket.terraform_state_bucket: Refreshing state... (ID: acme-tfstate-[redacted])

------------------------------------------------------------------------

No changes. Infrastructure is up-to-date.

This means that Terraform did not detect any differences between your
configuration and real physical resources that exist. As a result, no
actions need to be performed.
```

# Terraform in Action

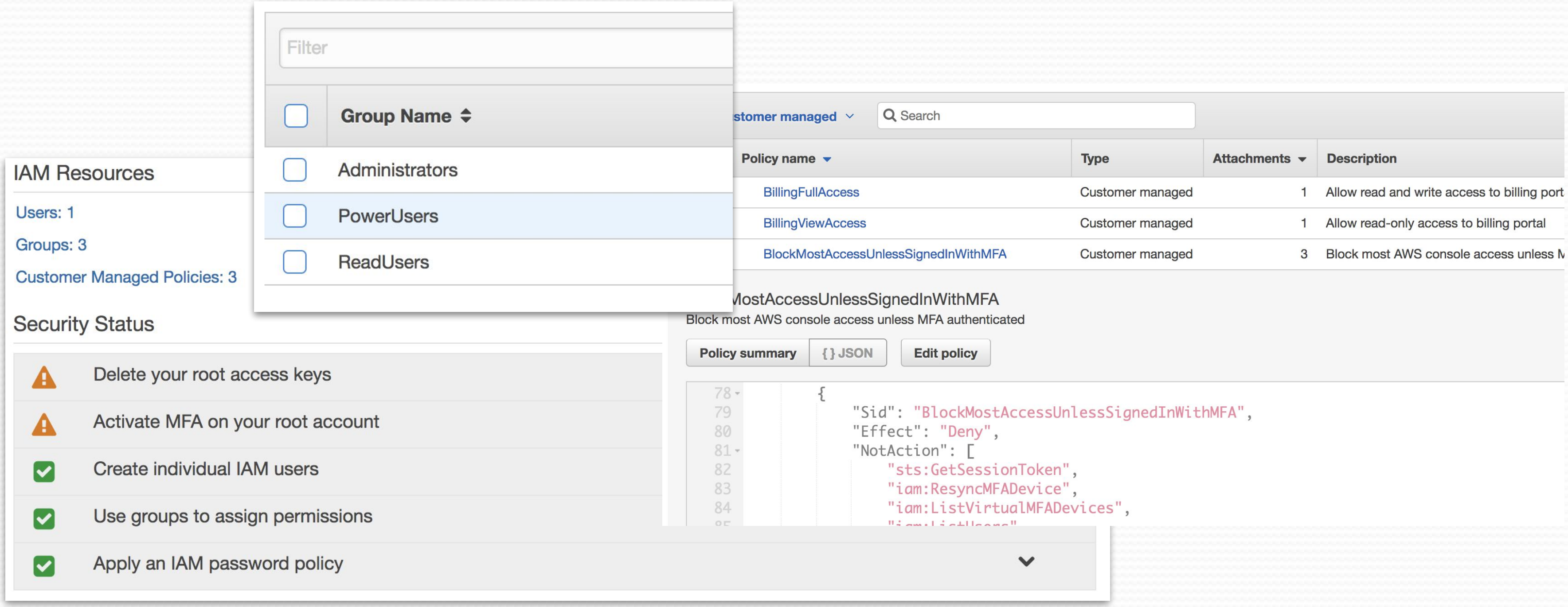
- Time to deploy to staging!
  - **Step 2:** account_security
  - Apply the changes (note - skipping plan for presentation for brevity)



```
module.iam_baseline_settings.aws_iam_policy.block_non_mfa_access_policy: Creation complete after 0s (ID: arn:aws:iam::[redacted]:p
olicy/BlockMostAccessUnlessSignedInWithMFA)
module.iam_baseline_settings.aws_iam_group_policy_attachment.readonlyusers_mfa_policy: Creating...
  group:      "" => "ReadUsers"
  policy_arn: "" => "arn:aws:iam::[redacted]:policy/BlockMostAccessUnlessSignedInWithMFA"
module.iam_baseline_settings.aws_iam_group_policy_attachment.powerusers_mfa_policy: Creating...
  group:      "" => "PowerUsers"
  policy_arn: "" => "arn:aws:iam::[redacted]:policy/BlockMostAccessUnlessSignedInWithMFA"
module.iam_baseline_settings.aws_iam_group_policy_attachment.administrator_mfa_policy: Creating...
  group:      "" => "Administrators"
  policy_arn: "" => "arn:aws:iam::[redacted]:policy/BlockMostAccessUnlessSignedInWithMFA"
module.iam_baseline_settings.aws_iam_account_password_policy.strict: Creation complete after 1s (ID: iam-account-password-policy)
module.iam_baseline_settings.aws_iam_group_policy_attachment.administrator_admin_policy: Creation complete after 1s (ID: Administrat
ors-[redacted])
module.iam_baseline_settings.aws_iam_group_policy_attachment.readonlyusers_readonly_policy: Creation complete after 1s (ID: ReadUser
s-[redacted])
module.iam_baseline_settings.aws_iam_group_policy_attachment.powerusers_billing_policy: Creation complete after 1s (ID: PowerUsers-[redacted]
[redacted])
module.iam_baseline_settings.aws_iam_group_policy_attachment.powerusers_power_policy: Creation complete after 1s (ID: PowerUsers-[redacted]
[redacted])
module.iam_baseline_settings.aws_iam_group_policy_attachment.administrator_billing_policy: Creation complete after 1s (ID: Administr
ators-[redacted])
module.iam_baseline_settings.aws_iam_group_policy_attachment.readonlyusers_mfa_policy: Creation complete after 1s (ID: ReadUsers-[redacted]
[redacted])
module.iam_baseline_settings.aws_iam_group_policy_attachment.powerusers_mfa_policy: Creation complete after 1s (ID: PowerUsers-[redacted]
[redacted])
module.iam_baseline_settings.aws_iam_group_policy_attachment.administrator_mfa_policy: Creation complete after 1s (ID: Administrator
s-[redacted])

Apply complete! Resources: 15 added, 0 changed, 0 destroyed.
```

# Terraform in Action

- Time to deploy to staging!
  - **Step 2:** account_security
  - Note that the account is now compliant with the desired security properties specified in the Terraform code:



IAM Resources

Users: 1

Groups: 3

Customer Managed Policies: 3

Security Status

- ⚠ Delete your root access keys
- ⚠ Activate MFA on your root account
- ✓ Create individual IAM users
- ✓ Use groups to assign permissions
- ✓ Apply an IAM password policy

Filter

| □ | Group Name |
|---|---|
| □ | Administrators |
| □ | PowerUsers |
| □ | ReadUsers |

stomer managed    Search

| Policy name | Type | Attachments | Description |
|---|---|---|---|
| BillingFullAccess | Customer managed | 1 | Allow read and write access to billing port |
| BillingViewAccess | Customer managed | 1 | Allow read-only access to billing portal |
| BlockMostAccessUnlessSignedInWithMFA | Customer managed | 3 | Block most AWS console access unless M |

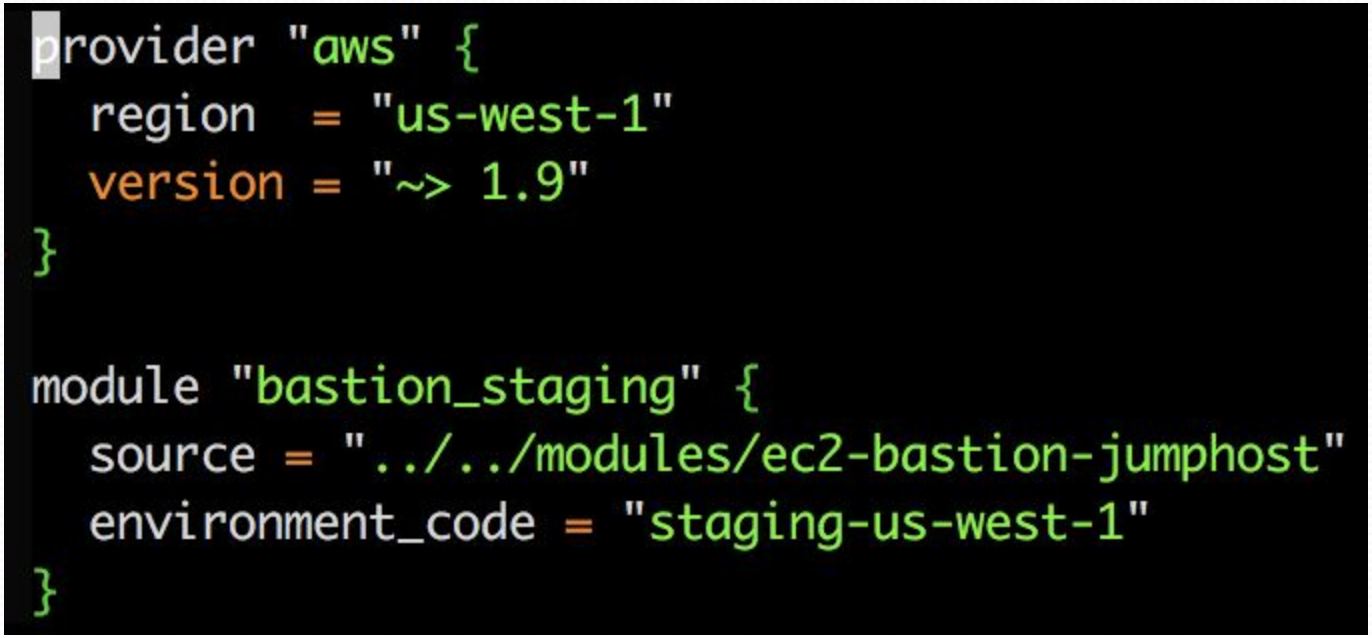
lostAccessUnlessSignedInWithMFA

Block most AWS console access unless MFA authenticated

```
        {
            "Sid": "BlockMostAccessUnlessSignedInWithMFA",
            "Effect": "Deny",
            "NotAction": [
                "sts:GetSessionToken",
                "iam:ResyncMFADevice",
                "iam:ListVirtualMFADevices",
```

# Terraform in Action

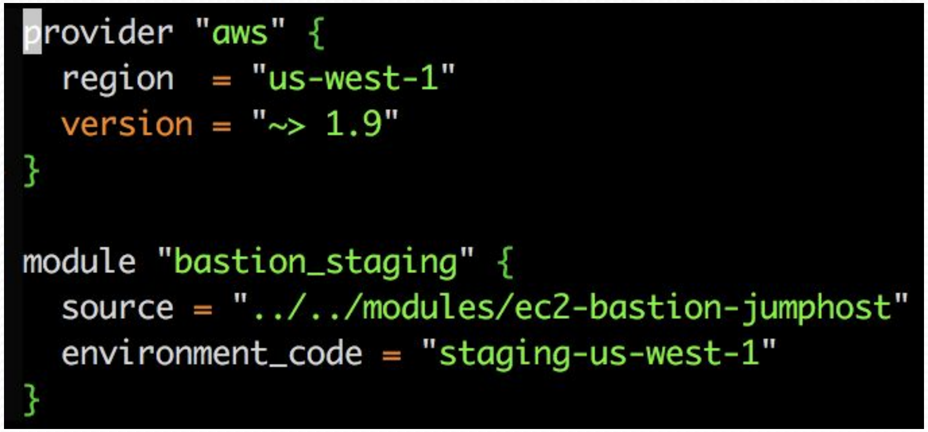
- Time to deploy to staging!
  - **Step 3:** bastion
  - **POP-QUIZ QUESTION**: Can you determine what the code below in the bastion environment module is going to do?



```
provider "aws" {
  region  = "us-west-1"
  version = "~> 1.9"
}

module "bastion_staging" {
  source = "../../modules/ec2-bastion-jumphost"
  environment_code = "staging-us-west-1"
}
```

# Terraform in Action

- Time to deploy to staging!
  - **Step 3:** bastion
  - **ANSWER**: It is deploying a feature module called "ec2-bastion-jumphost".

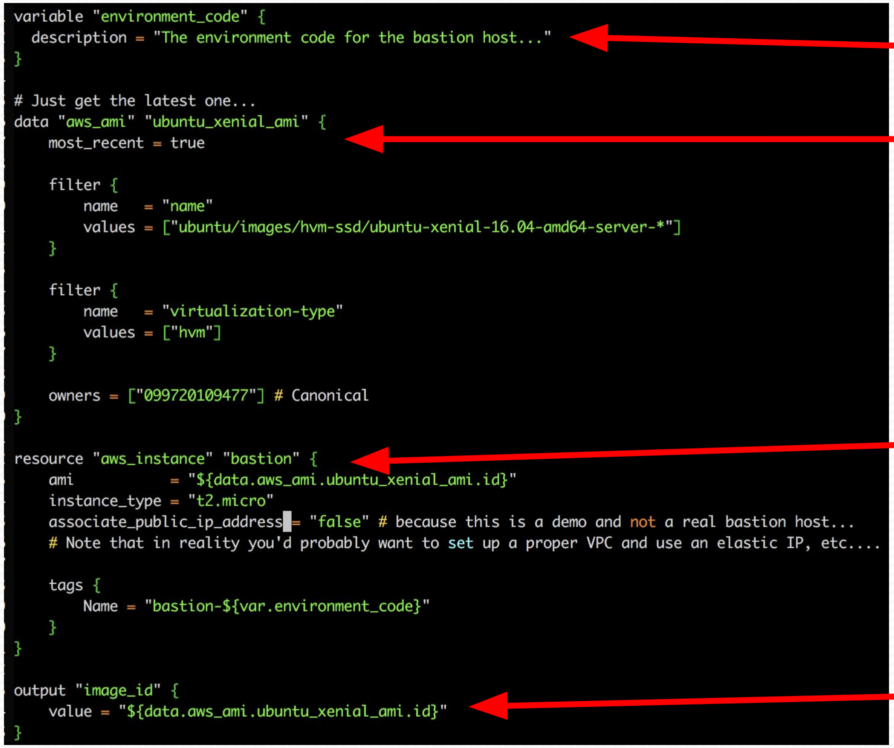


```
provider "aws" {
  region  = "us-west-1"
  version = "~> 1.9"
}

module "bastion_staging" {
  source = "../../modules/ec2-bastion-jumphost"
  environment_code = "staging-us-west-1"
}
```

  - **QUESTION**: What is ec2-bastion-jumphost going to do? Let's take a look at the module ec2-bastion-jumphost's definition as shown by the code.

# Terraform in Action

- Time to deploy to staging!
  - **Step 3:** bastion
  - **QUESTION**: What is ec2-bastion-jumphost going to do?



```
variable "environment_code" {
  description = "The environment code for the bastion host..."
}

# Just get the latest one...
data "aws_ami" "ubuntu_xenial_ami" {
    most_recent = true

    filter {
        name   = "name"
        values = ["ubuntu/images/hvm-ssd/ubuntu-xenial-16.04-amd64-server-*"]
    }

    filter {
        name   = "virtualization-type"
        values = ["hvm"]
    }

    owners = ["099720109477"] # Canonical
}

resource "aws_instance" "bastion" {
    ami           = "${data.aws_ami.ubuntu_xenial_ami.id}"
    instance_type = "t2.micro"
    associate_public_ip_address = "false" # because this is a demo and not a real bastion host...
    # Note that in reality you'd probably want to set up a proper VPC and use an elastic IP, etc....

    tags {
        Name = "bastion-${var.environment_code}"
    }
}

output "image_id" {
    value = "${data.aws_ami.ubuntu_xenial_ami.id}"
}
```

Require a variable called "environment code"

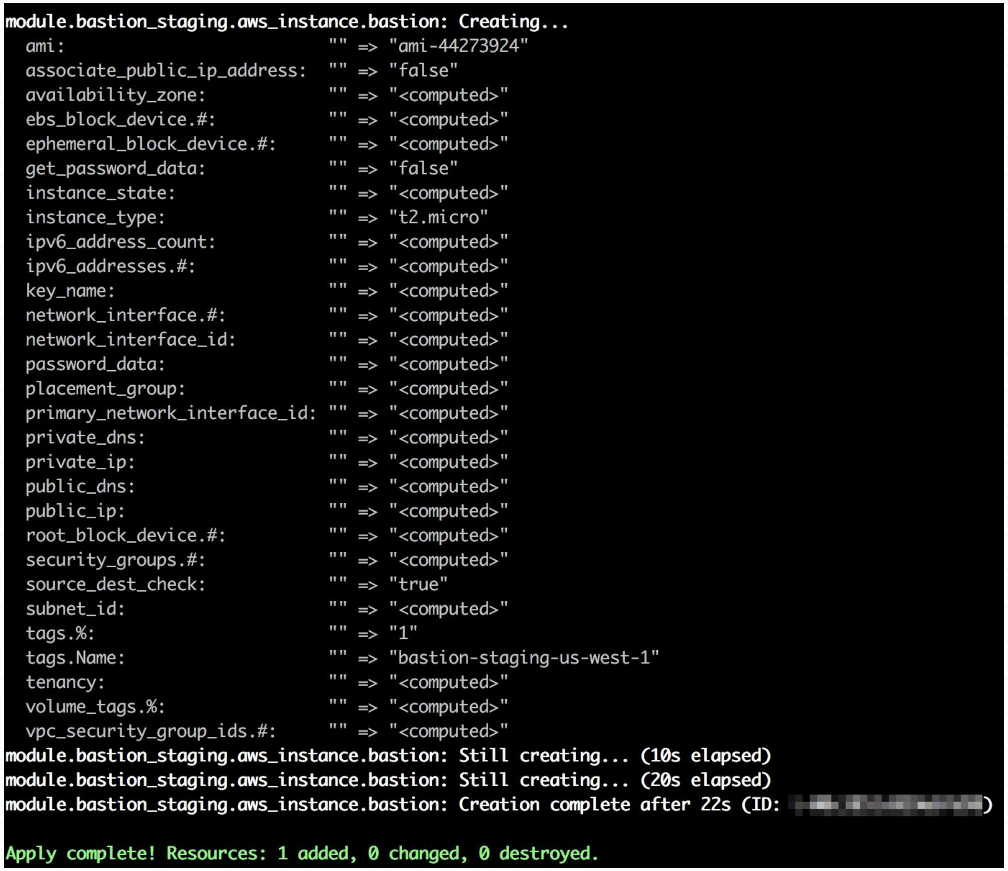
Figure out the latest Ubuntu Xenial image from upstream vendor Canonical (the organization that curates the Ubuntu Linux distribution)

Deploy an AWS EC2 instance called "bastion-${var.environment_code}" where ${var.environment_code} will interpolate to the variable environment_code passed in.

Provide an output of the AMI used, storable and queryable via Terraform state.

# Terraform in Action

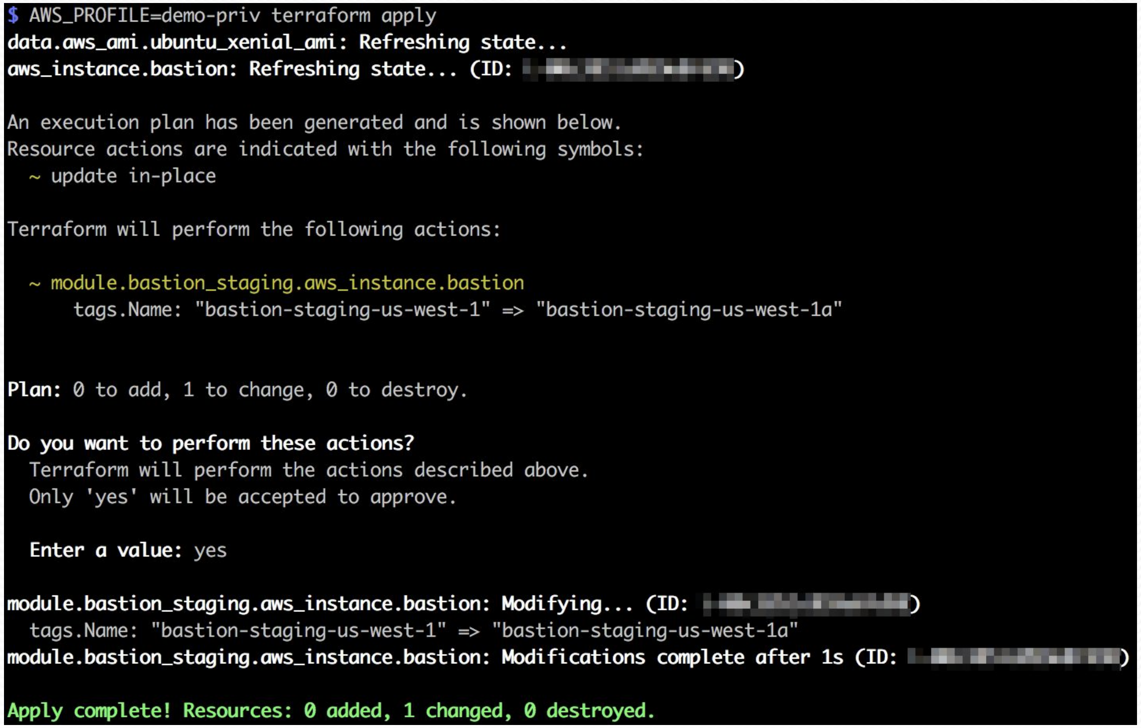
- Time to deploy to staging!
  - **Step 3:** bastion
  - Deploy it!
    - The AMI is found based on the data resource in the feature module
    - The EC2 instance is deployed based on the resource defined in the feature module.



```
module.bastion_staging.aws_instance.bastion: Creating...
  ami:                          "" => "ami-44273924"
  associate_public_ip_address:  "" => "false"
  availability_zone:            "" => "<computed>"
  ebs_block_device.#:           "" => "<computed>"
  ephemeral_block_device.#:     "" => "<computed>"
  get_password_data:            "" => "false"
  instance_state:               "" => "<computed>"
  instance_type:                "" => "t2.micro"
  ipv6_address_count:           "" => "<computed>"
  ipv6_addresses.#:             "" => "<computed>"
  key_name:                     "" => "<computed>"
  network_interface.#:          "" => "<computed>"
  network_interface_id:         "" => "<computed>"
  password_data:                "" => "<computed>"
  placement_group:              "" => "<computed>"
  primary_network_interface_id: "" => "<computed>"
  private_dns:                  "" => "<computed>"
  private_ip:                   "" => "<computed>"
  public_dns:                   "" => "<computed>"
  public_ip:                    "" => "<computed>"
  root_block_device.#:          "" => "<computed>"
  security_groups.#:            "" => "<computed>"
  source_dest_check:            "" => "true"
  subnet_id:                    "" => "<computed>"
  tags.%:                       "" => "1"
  tags.Name:                    "" => "bastion-staging-us-west-1"
  tenancy:                      "" => "<computed>"
  volume_tags.%:                "" => "<computed>"
  vpc_security_group_ids.#:     "" => "<computed>"
module.bastion_staging.aws_instance.bastion: Still creating... (10s elapsed)
module.bastion_staging.aws_instance.bastion: Still creating... (20s elapsed)
module.bastion_staging.aws_instance.bastion: Creation complete after 22s (ID: [redacted])

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

# Terraform in Action

- Time to deploy to staging!
  - **Step 3:** bastion
  - Oops!
  - The tag "name" should have been set to: bastion-staging-us-west-1a
    - Update the tag and try re-applying.
    - Note that TF tracks and applies change.



```
$ AWS_PROFILE=demo-priv terraform apply
data.aws_ami.ubuntu_xenial_ami: Refreshing state...
aws_instance.bastion: Refreshing state... (ID: [redacted])

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  ~ update in-place

Terraform will perform the following actions:

  ~ module.bastion_staging.aws_instance.bastion
      tags.Name: "bastion-staging-us-west-1" => "bastion-staging-us-west-1a"


Plan: 0 to add, 1 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

module.bastion_staging.aws_instance.bastion: Modifying... (ID: [redacted])
  tags.Name: "bastion-staging-us-west-1" => "bastion-staging-us-west-1a"
module.bastion_staging.aws_instance.bastion: Modifications complete after 1s (ID: [redacted])

Apply complete! Resources: 0 added, 1 changed, 0 destroyed.
```

# Terraform in Action

- Deployment is complete!
  - This can be repeated across multiple environments, and the result is well defined and determinate.
  - Allows infrastructure changes to be rolled forward and back based on code commits/releases from an SCM.
  - Allows infrastructure to be change managed and tracked in SCM, just like any other code.
  - Also allows easy teardown of AWS resources, i.e. terraform destroy…
  - You can query and introspect the Terraform state to determine which objects in AWS were created outside of the formal project structure.

# Quick Words on Packer

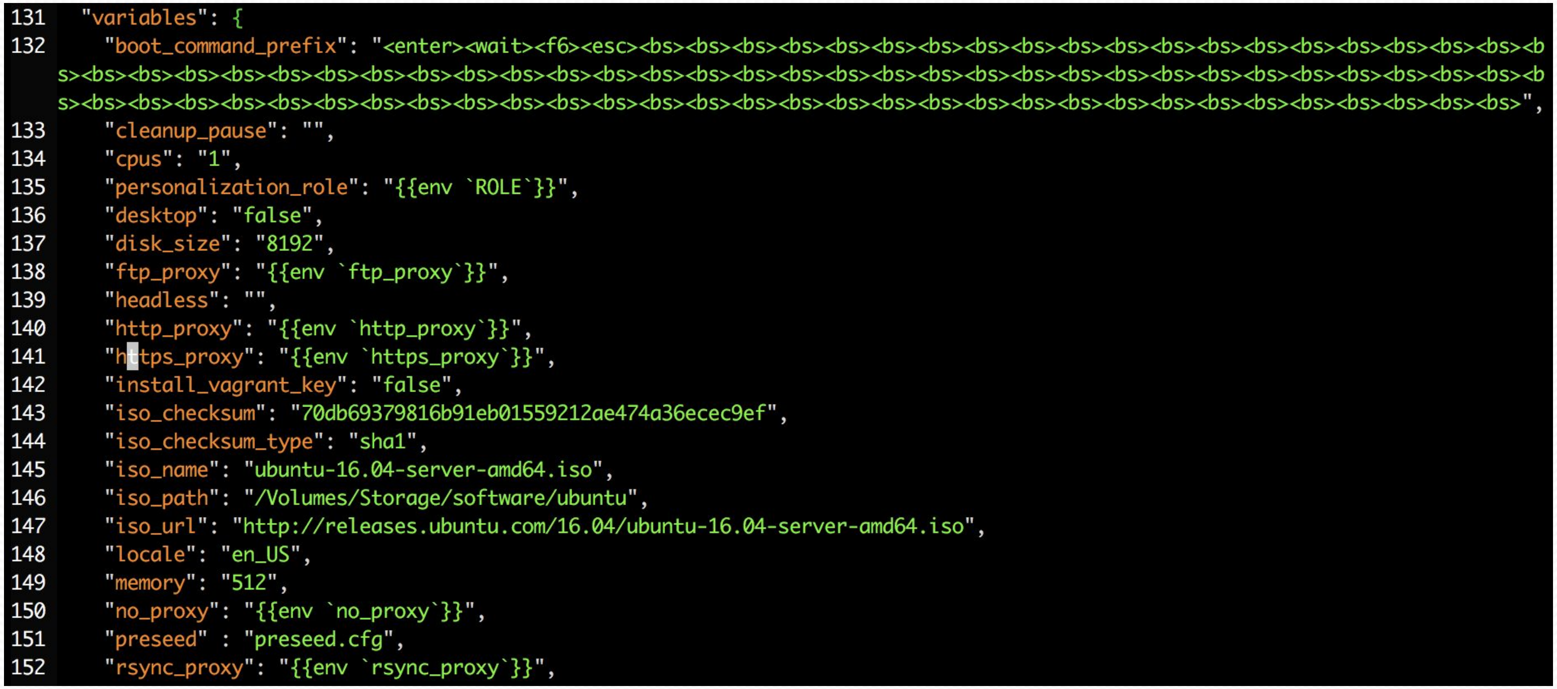
- Packer allows creation and/or customization of virtual machine images
  - In other words, it allows custom AMIs to be built in the context of the AWS environment
    - But supports many virtualization hosts, which is useful
      - Can use a local VM host to build images for the cloud based on validated ISOs from upstream vendors
        - Might be important if you are operating in sensitive environments that don't let you access public images
        - Do you trust the images in the marketplace?
    - From the same company as Terraform, shares a lot of similar configuration principles.
    - Pro-tip: Make your life easy by adapting pre-existing recipes via projects like Boxcutter.

# Quick Words on Packer

- Using Packer, tools like cloud-init and other requirements can be pre-configured as desired.
  - Default security configurations in the OS?
  - Ability to burn specific AMIs for different functionality or roles if desired (or not):
    - with pre-loaded components "burned" into the AMI (vs. runtime configuration at instantiation); or
    - by using scripts to configure during instantiation, which allows EC2 metadata and APIs to be queried and change configuration of image at point of deployment.
      - Pros and cons to both techniques; feel free to use one, the other, or both methodologies as needed.

# Quick Words on Packer

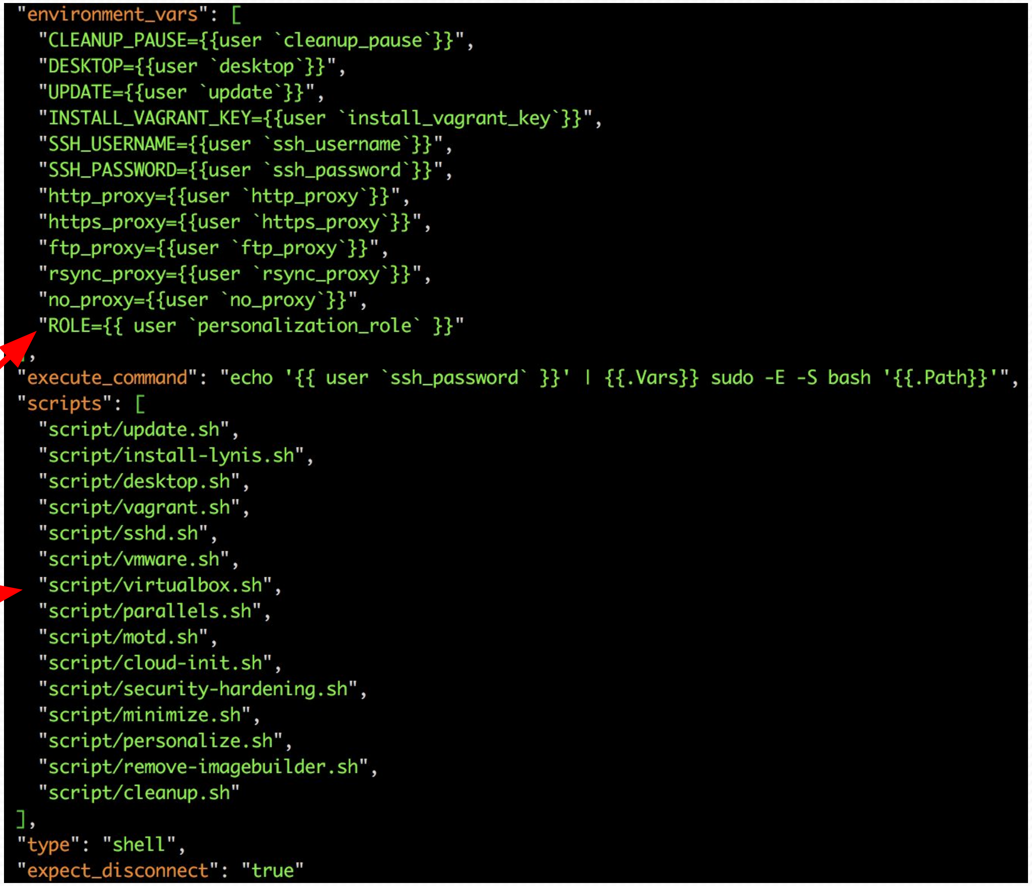
- Sample stanza showing variables in a packer configuration (from a heavily customized fork of the Boxcutter Ubuntu project):



```
  "variables": {
    "boot_command_prefix": "<enter><wait><f6><esc><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs><bs>",
    "cleanup_pause": "",
    "cpus": "1",
    "personalization_role": "{{env `ROLE`}}",
    "desktop": "false",
    "disk_size": "8192",
    "ftp_proxy": "{{env `ftp_proxy`}}",
    "headless": "",
    "http_proxy": "{{env `http_proxy`}}",
    "https_proxy": "{{env `https_proxy`}}",
    "install_vagrant_key": "false",
    "iso_checksum": "70db69379816b91eb01559212ae474a36ecec9ef",
    "iso_checksum_type": "sha1",
    "iso_name": "ubuntu-16.04-server-amd64.iso",
    "iso_path": "/Volumes/Storage/software/ubuntu",
    "iso_url": "http://releases.ubuntu.com/16.04/ubuntu-16.04-server-amd64.iso",
    "locale": "en_US",
    "memory": "512",
    "no_proxy": "{{env `no_proxy`}}",
    "preseed" : "preseed.cfg",
    "rsync_proxy": "{{env `rsync_proxy`}}",
```

# Quick Words on Packer

- In this example, note that we pass an environment variable ROLE during packer execution, passed upstream by the caller
- This is used by various scripts to support multiple customizations from a single reusable codebase.



```
"environment_vars": [
  "CLEANUP_PAUSE={{user `cleanup_pause`}}",
  "DESKTOP={{user `desktop`}}",
  "UPDATE={{user `update`}}",
  "INSTALL_VAGRANT_KEY={{user `install_vagrant_key`}}",
  "SSH_USERNAME={{user `ssh_username`}}",
  "SSH_PASSWORD={{user `ssh_password`}}",
  "http_proxy={{user `http_proxy`}}",
  "https_proxy={{user `https_proxy`}}",
  "ftp_proxy={{user `ftp_proxy`}}",
  "rsync_proxy={{user `rsync_proxy`}}",
  "no_proxy={{user `no_proxy`}}",
  "ROLE={{ user `personalization_role` }}"
],
"execute_command": "echo '{{ user `ssh_password` }}' | {{.Vars}} sudo -E -S bash '{{.Path}}'",
"scripts": [
  "script/update.sh",
  "script/install-lynis.sh",
  "script/desktop.sh",
  "script/vagrant.sh",
  "script/sshd.sh",
  "script/vmware.sh",
  "script/virtualbox.sh",
  "script/parallels.sh",
  "script/motd.sh",
  "script/cloud-init.sh",
  "script/security-hardening.sh",
  "script/minimize.sh",
  "script/personalize.sh",
  "script/remove-imagebuilder.sh",
  "script/cleanup.sh"
],
"type": "shell",
"expect_disconnect": "true"
```

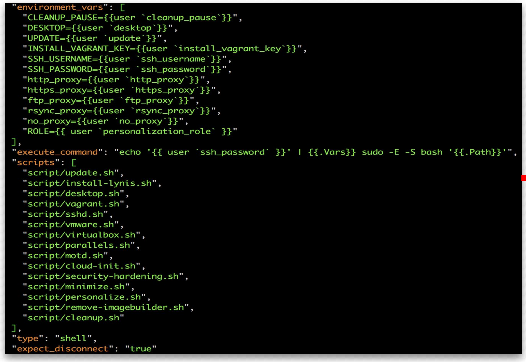
# Quick Words on Packer

- Depending on whether the source is a bootable ISO or an existing machine image (AMI in this case), Packer will:
  - For an ISO image
    - Spin up a VM using your VM host of choice (i.e. Virtualbox, VMWare, etc.)
    - For Linux, use preseeding or kickstart to perform headless/auto install using specified configuration.
    - Perform customization/personalization steps
    - Halt and export the VM image for import to AWS.
  - For an existing AMI
    - Spin up EC2 instance using specified upstream AMI
    - Perform customization/personalization steps
    - Halt and dump the EC2 machine to an AMI image

# Quick Words on Packer

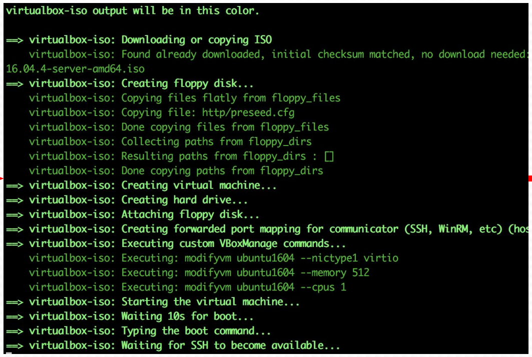
- Visual Overview of Packer's inputs and outputs:

1. Provide definition/config



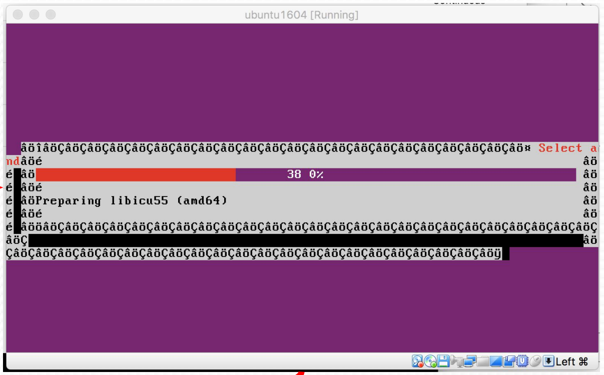
```
"environment_vars": [
  "CLEANUP_PAUSE={{user `cleanup_pause`}}",
  "DESKTOP={{user `desktop`}}",
  "UPDATE={{user `update`}}",
  "INSTALL_VAGRANT_KEY={{user `install_vagrant_key`}}",
  "SSH_USERNAME={{user `ssh_username`}}",
  "SSH_PASSWORD={{user `ssh_password`}}",
  "http_proxy={{user `http_proxy`}}",
  "https_proxy={{user `https_proxy`}}",
  "ftp_proxy={{user `ftp_proxy`}}",
  "rsync_proxy={{user `rsync_proxy`}}",
  "no_proxy={{user `no_proxy`}}",
  "ROLE={{ user `personalization_role` }}"
],
"execute_command": "echo '{{ user `ssh_password` }}' | {{.Vars}} sudo -E -S bash '{{.Path}}'",
"scripts": [
  "script/update.sh",
  "script/install-lynis.sh",
  "script/desktop.sh",
  "script/vagrant.sh",
  "script/sshd.sh",
  "script/vmware.sh",
  "script/virtualbox.sh",
  "script/parallels.sh",
  "script/motd.sh",
  "script/cloud-init.sh",
  "script/security-hardening.sh",
  "script/minimize.sh",
  "script/personalize.sh",
  "script/remove-imagebuilder.sh",
  "script/cleanup.sh"
],
"type": "shell",
"expect_disconnect": "true"
```

2. Packer spins up VM+instrumentation

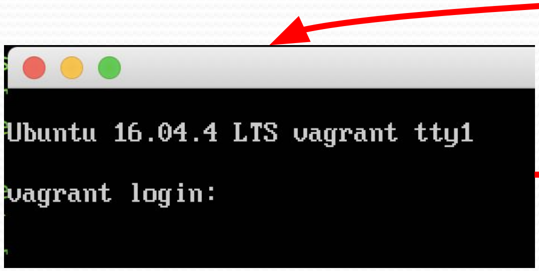


```
virtualbox-iso output will be in this color.

==> virtualbox-iso: Downloading or copying ISO
    virtualbox-iso: Found already downloaded, initial checksum matched, no download needed:
16.04.4-server-amd64.iso
==> virtualbox-iso: Creating floppy disk...
    virtualbox-iso: Copying files flatly from floppy_files
    virtualbox-iso: Copying file: http/preseed.cfg
    virtualbox-iso: Done copying files from floppy_files
    virtualbox-iso: Collecting paths from floppy_dirs
    virtualbox-iso: Resulting paths from floppy_dirs : []
    virtualbox-iso: Done copying paths from floppy_dirs
==> virtualbox-iso: Creating virtual machine...
==> virtualbox-iso: Creating hard drive...
==> virtualbox-iso: Attaching floppy disk...
==> virtualbox-iso: Creating forwarded port mapping for communicator (SSH, WinRM, etc) (hos
==> virtualbox-iso: Executing custom VBoxManage commands...
    virtualbox-iso: Executing: modifyvm ubuntu1604 --nictype1 virtio
    virtualbox-iso: Executing: modifyvm ubuntu1604 --memory 512
    virtualbox-iso: Executing: modifyvm ubuntu1604 --cpus 1
==> virtualbox-iso: Starting the virtual machine...
==> virtualbox-iso: Waiting 10s for boot...
==> virtualbox-iso: Typing the boot command...
==> virtualbox-iso: Waiting for SSH to become available...
```

3. The operating system is installed in VM

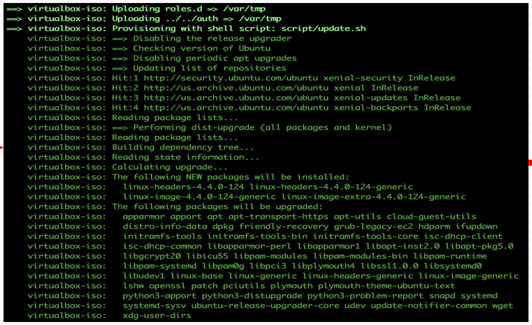


```
 âöîâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâö¤ Select a
ndâöé                                                                  âö
é âö                          38 0%                                    âö
é âöé                                                                  âö
é âöPreparing libicu55 (amd64)                                          âö
é âöé                                                                  âö
é âööâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇ
âöÇ                                                                    âö
ÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÇâöÿ
```



```
Ubuntu 16.04.4 LTS vagrant tty1

vagrant login:
```

4. Boot into the new VM once installed



```
==> virtualbox-iso: Uploading roles.d => /var/tmp
==> virtualbox-iso: Uploading ../../auth => /var/tmp
==> virtualbox-iso: Provisioning with shell script: script/update.sh
    virtualbox-iso: ==> Disabling the release upgrader
    virtualbox-iso: ==> Checking version of Ubuntu
    virtualbox-iso: ==> Disabling periodic apt upgrades
    virtualbox-iso: ==> Updating list of repositories
    virtualbox-iso: Hit:1 http://security.ubuntu.com/ubuntu xenial-security InRelease
    virtualbox-iso: Hit:2 http://us.archive.ubuntu.com/ubuntu xenial InRelease
    virtualbox-iso: Hit:3 http://us.archive.ubuntu.com/ubuntu xenial-updates InRelease
    virtualbox-iso: Hit:4 http://us.archive.ubuntu.com/ubuntu xenial-backports InRelease
    virtualbox-iso: Reading package lists...
    virtualbox-iso: ==> Performing dist-upgrade (all packages and kernel)
    virtualbox-iso: Reading package lists...
    virtualbox-iso: Building dependency tree...
    virtualbox-iso: Reading state information...
    virtualbox-iso: Calculating upgrade...
    virtualbox-iso: The following NEW packages will be installed:
    virtualbox-iso:   linux-headers-4.4.0-124 linux-headers-4.4.0-124-generic
    virtualbox-iso:   linux-image-4.4.0-124-generic linux-image-extra-4.4.0-124-generic
    virtualbox-iso: The following packages will be upgraded:
    virtualbox-iso:   apparmor apport apt apt-transport-https apt-utils cloud-guest-utils
    virtualbox-iso:   distro-info-data dpkg friendly-recovery grub-legacy-ec2 hdparm ifupdown
    virtualbox-iso:   initramfs-tools initramfs-tools-bin initramfs-tools-core isc-dhcp-client
    virtualbox-iso:   isc-dhcp-common libapparmor-perl libapparmor1 libapt-inst2.0 libapt-pkg5.0
    virtualbox-iso:   libgcrypt20 libicu55 libpam-modules libpam-modules-bin libpam-runtime
    virtualbox-iso:   libpam-systemd libpam0g libpci3 libplymouth4 libssl1.0.0 libsystemd0
    virtualbox-iso:   libudev1 linux-base linux-generic linux-headers-generic linux-image-generic
    virtualbox-iso:   lshw openssl patch pciutils plymouth plymouth-theme-ubuntu-text
    virtualbox-iso:   python3-apport python3-distupgrade python3-problem-report snapd systemd
    virtualbox-iso:   systemd-sysv ubuntu-release-upgrader-core udev update-notifier-common wget
    virtualbox-iso:   xdg-user-dirs
```

5. Packer customizes the new VM

Machine Image

6. Packer dumps/exports to new image

# Results...

- GIT repository access controls are used to enforce a workflow that mandates code reviews and allows deployments to occur from the certified/approved release branches.
- Terraform is used to deploy infrastructure based on code from the certified/approved release branches
  - Terraform also provides us with some useful change management features, like showing what's going to happen before it does.
- Packer is used to build machine images based on code from the certified/approved release branches.
- It is now possible to demonstrate the pipeline for compliance purposes.

# Results…

- Some useful compliance attributes:
  - Infrastructure assets are now tracked across their lifecycle via Terraform state.
  - Infrastructure assets are directly related to the definition(s) that live in the SCM and Terraform state.
  - Machine images can be signed, validated, and have a known footprint.
    - Configuration is consistent across builds and does not rely on manual intervention.
  - If infrastructure is designed to be immutable, upgrades become a matter of generating and deploying new images. Eliminates risk of drift.
  - Infrastructure can be (more?) easily re-deployed for BC/DR

# Questions…

- Q&A

- Resource list:
  - Packer: https://www.packer.io/
  - Terraform: https://www.terraform.io/
  - GIT: https://git-scm.com/
    - Noted GIT tools:
      - Gitolite: http://gitolite.com/gitolite/index.html
      - Gerrit: https://www.gerritcodereview.com/
      - GOGS: https://www.gogs.io/
      - Gitlab: https://about.gitlab.com/
      - Github: https://www.github.com/ (proprietary/closed source)

# Thank you

**Stuart Cianos, CISSP**
scianos@alphavida.com
Security Architect @ Medallia

**Disclaimer**

The views and opinions expressed during this conference are those of the speakers and do not necessarily reflect the views and opinions held by the Information Systems Security Association (ISSA), the Silicon Valley ISSA, the San Francisco ISSA or the San Francisco Bay Area InfraGard Members Alliance (IMA). Neither ISSA, InfraGard, nor any of its chapters warrants the accuracy, timeliness or completeness of the information presented. Nothing in this conference should be construed as professional or legal advice or as creating a professional-customer or attorney-client relationship. If professional, legal, or other expert assistance is required, the services of a competent professional should be sought.